Prepared by: Victor_TheOracle

Lead Auditors:

- Victor_TheOracle

Table of Contents

Protocol Summary

PasswordStore is a protocol that stores and retrieves a user's password. The protocol is deigned to e used by one user only. Only the owner should be able to access the functions here.

Disclaimer

I, Victor_TheOracle, have made all effort to find as many vulnerabilities in the code in the given time period, but hold no responsibilities for the findings provided in this document. A security audit by me is not an endorsement of the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the Solidity implementation of the contracts.

Risk Classification

	Impact		
	High	Medium	Low

		Impact		
	High	H	H/M	M
Likelihood	Medium	H/M	M	M/L
	Low	M	M/L	L

I use the CodeHawks severity matrix to determine severity. See the documentation for more details.

Audit Details

The findings desribed in this document correspond to the following commit hash:

```
2e8f81e263b3a9d18fab4fb5c46805ffc10a9990
```

Scope

```
./src/
└── PasswordStore.sol
```

Roles

- Owner: Is the only one who should be able to set and access the password.

Executive Summary

Issues found

Severity	Number of issues found
High	2
Low	1
Info	1
Total	4

Findings

High

[H-1] Storing the password on-chain makes it visible to anyone, hence, it is no longer private.

Description: All the data stored on-chain is visible to anyone, and can be read directly from the blockchain. The `PasswordStore::s_password` variable is intended to be a private variable and only accessed through the `PasswordStore::getPAssword` function, which is intended to only be called by the owner of the contract.

Impact: Anyone can read the private password. This defeats the entire purpose of the protocol.

Proof of Concept: (Proof of Code)

The below test case shoows how anyone can read the password directly from the blockchain.

1. Create a locally running chain

```
make anvil
```

2. Deploy the contract to the chain

```
make deploy
```

3. Run the storage tool

We use `1` because that is the storage slot of `s_password` in the contract.

```
cast storage <ADDRESS_HERE> 1 --rpc-url http://127.0.0.1:8545
```

You will get an output similar to this
`0x6d7950617373776f726400000000000000000000000000000000000000000014`

Proceed to parse the hex to a string with:

```
cast parse-bytes32-string
0x6d7950617373776f726400000000000000000000000000000000000000000014
```

And get an output of:

```
myPassword
```

Recommended Mitigation: The entire architecture of this contract should be rethought. One could encrypt the password off-chain, and then store he encrypted password on-chain. This would require the user to remember another password off-chain to decrypt the password. However, you'd also likely want to remove the view function as you would not want the user to accidentally send a transaction with the password that encrypts your password.

[H-2] `PasswordStore::setPassword` has no access controls. This means a non-owner can change the stored password.

Informational

[I-1] The `PasswordStore::getPassword` nastspec indicates a parameter that does not exist

Description:

```
    /*
     * @notice This allows only the owner to retrieve the password.
     * @param newPassword The new password to set.
     */
    function getPassword() external view returns (string memory) {
        if (msg.sender != s_owner) {
            revert PasswordStore__NotOwner();
        }
        return s_password;
    }
```

The natspec imples that it should be `getPassword(string)`

Impact: The natspec is incorrect

Recommended Mitigation: Remove the incorrect natspec line

```
- * @param newPassword The new password to set.
```